VNE-Sim: A Virtual Network Embedding Simulator

Soroush Haeri and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
{shaeri, ljilja}@sfu.ca

ABSTRACT

Computer network virtualization and virtual network embedding (VNE) algorithms have gained significant attention in recent years. Performance evaluation of VNE algorithms mainly relies on discrete event simulations. However, modular and scalable VNE simulators are not available. In this paper, we describe newly developed simulator *VNE-Sim* that enables users to define and implement network elements possessing various resources. It also enables researchers to generate various topologies of Internet Service Provider and data center networks for the evaluation of VNE algorithms. Furthermore, *VNE-Sim* provides the infrastructure for implementing both single and batch request processing approaches.

CCS Concepts

•Networks → Network simulations; •Computing methodologies → Simulation environments; Simulation tools; •Software and its engineering → *Abstraction, modeling and modularity;*

Keywords

Computer networks, virtualization, virtual network embedding, simulation tools, discrete event simulations

1. INTRODUCTION

The best-effort service, supported by the current Internet architecture, is not well-suited for all applications. A significant barrier to innovation has been imposed by the inability of the current Internet architecture to support a diverse array of applications. Network virtualization overcomes these shortcomings [1], [2]. The virtualized network model divides the role of Internet Service Providers (ISPs) into two independent entities: Infrastructure Providers (InPs) and Service Providers (SPs). The InPs manage the physical infrastructure while the SPs aggregate resources from multiple InPs into multiple Virtual Networks (VNs) to provide end-to-end services [2]. In the virtualized network architecture, an InP owns and operates a *substrate network* composed of physical nodes and links that are interconnected. Combinations of the substrate network nodes and links may be used to embed virtualized networks.

An InP's revenue depends on the resource utilization within the substrate network that, in turn, depends on the performance of the algorithm that allocates the substrate network resources to virtual networks. This resource allocation is known as the virtual network embedding (VNE) [3], which may be formulated as a mixed-integer program (MIP) [4] or may be reduced to the multiway separator problem [3], [5]. Both problems are $\mathcal{NP}$-hard, making the VNE problem also $\mathcal{NP}$-hard. This is one of the main challenges of virtualization.

VNE algorithms have recently received considerable attention within the research community [3], [4], [6]. Performance of VNE algorithms is only evaluated using discrete event simulations. Therefore, there is a need for a scalable, flexible, reliable, and modular VNE simulator.

Embed [3] and *ViNEYard* [4] are two early developed C-based VNE simulators. They are no longer maintained, lack documentation, and do not provide interfaces for developing new VNE algorithms and performance metrics. MATLAB has also been used for simulating VNE algorithms [6]. However, the MATLAB-based simulator is not publicly available. *Alevin* [7] is a recent VNE simulator written in Java. It has a modular design and provides a graphical user interface (GUI). However, its design does not enable users to define network elements of their choice such as defining CPU, memory, and storage of nodes and bandwidth of links in the substrate network. Furthermore, writing scripts to perform batch simulations is rather cumbersome.

We have developed the *VNE-Sim*, a virtual network embedding simulator that has been used for performance evaluation of a number of VNE algorithms [8], [9]. It is publicly available under the terms of the MIT License. *VNE-Sim* provides extensible interfaces for users to implement algorithms and customizable network elements. Furthermore, being written in C++, *VNE-Sim* offers good memory management and enables performing batch simulations using any scripting language.

The remainder of this paper is organized as follows. The *VNE-Sim* architecture and its components are introduced in Section 2. The core of the simulator is then presented in Section 3. In Section 4, we present a case study and introduce the VNE problem, common approaches for its solutions, and simulation results. We conclude with Section 5.

.

2. THE VNE-SIM ARCHITECTURE

VNE-Sim is a discrete event simulator written using the 2011 C++ standard (C++11), which introduces smart pointers and lambda functions. These new functionalities of C++11 are employed in *VNE-Sim*. We describe here details of the *VNE-Sim* discrete event simulator and its compilation. The *CMake* build system [10] is employed for compiling the code. *VNE-Sim* relies on several external libraries. While some required libraries are downloaded and installed automatically by *CMake*, the others libraries are expected to be installed prior to initiating the compilation process.

The libraries that are automatically installed by *CMake* are: *Fast Network Simulation Setup* (FNSS) [11] used for generating data center network topologies; the *Hiberlite* library [12] used for writing simulation results to disk; and the *Adevs* library [13] employed for modeling the virtual network embedding process as a discrete event system.

CMake relies on scripts to search for the libraries that should be installed prior to compilation:

- The Boost File System, Log, Thread, and Unit Test Framework libraries [14] are mainly used in the core classes for file handling, logging, testing, and debugging. The test cases and experiments are implemented using the Unit Test Framework library.
- The GNU Scientific Library (GSL) [15] is used for generating random numbers and probability distributions.
- The GNU Linear Programming Kit (GLPK) [16] is used for solving the Multi-Commodity Flow (MCF) [5] problem. It is also employed by the ViNE algorithms [4].
- The SQLite3 library [17] handles simulation results. They are exported automatically as SQLite databases.

VNE-Sim uses modified versions of the *Hiberlite* and FNSS libraries. *CMake* applies the required modifications after downloading these libraries. *VNE-Sim* also contains a modified version of the Boston University Representative Topology Generator (BRITE) [18] that is compiled as a component of the simulator.

Components of *VNE-Sim* and their dependencies are shown in Figure 1.

The simulator includes a number of packages:

- The *Core* package contains classes and interfaces required for implementing various virtual network embedding algorithms. It also contains the discrete event simulation system that models the process of embedding virtual networks as a discrete event system.
- The *network-generator* package is used for generating various network topologies and Virtual Network Requests (VNRs). It employs the FNSS [11] and BRITE [18] libraries for generating network topologies and the GNU Scientific Library [15] for generating probablility distributions of VNR arrival times, lifetimes, and resource requirements.
- The *GRC* package contains the implementation of the Global Resource Capacity (GRC) algorithm [6].
- The *ViNEYard* package contains the implementation of R-ViNE and D-ViNE algorithms [4].

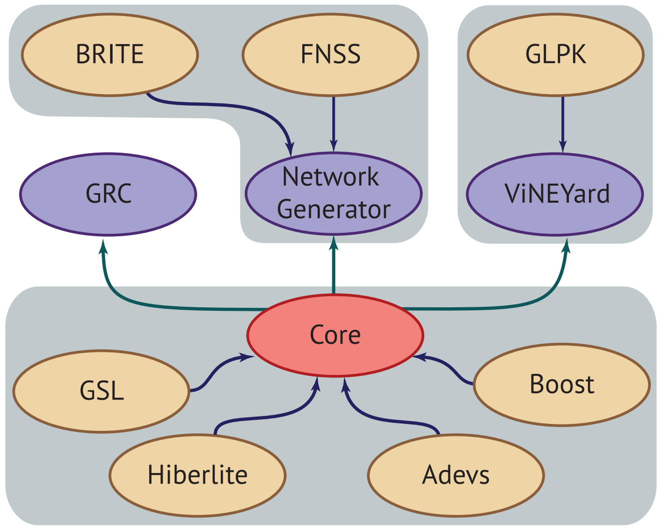

Figure 1: *VNE-Sim* components and their dependencies. Installation of Boost, GSL, and GLPK is required by the user while the other libraries are automatically installed during the compilation process.

3. THE VNE-SIM CORE

Components required to simulate various VNE algorithms are implemented as abstract template classes in the *Core* package. They are divided into five categories:

- *Network Component Classes* provide a framework for defining substrate and virtual nodes and links, substrate and virtual networks, and VNRs.
- *Virtual Network Embedding Algorithm Classes* define an interface for embedding algorithms.
- *Discrete Event Simulation Classes* model the virtual network embedding process as a discrete event system. All these classes are derived from the *Adevs* library [13].
- *Experiment and Result Collection Classes* connect various components to define simulation scenarios. They create SQLite database tables for simulation parameters and for collecting simulation results.
- *Operation Related Classes* provide the basic required functionalities such as managing the configuration file, database access, type definitions, and random number generation.

3.1 Network Component Classes

3.1.1 Substrate and Virtual Nodes and Links

Substrate and virtual network components are implemented as C++ *variadic* templates where their resources are defined by the template arguments. This design enables users to define network components that possess multiple resources. While the VNE algorithms often consider CPU processing capacity as the only node resource, other resources such as Random Access Memory (RAM) and storage capacity may also be provided by SPs. Hence, VNE algorithms should also consider these resources.

3.1.2 Networks and Virtual Network Requests

A network in *VNE-Sim* is defined by the `Network` template class with a node and a link class as its first and second arguments, respectively. This approach enables implementing substrate and virtual networks using one base template class. VNRs are also implemented as template classes where the template argument identifies the type of the virtual network object they possess.

3.2 Virtual Network Embedding Algorithm Classes

These classes define an interface for implementing VNE algorithms. Their backbone is the `EmbeddingAlgorithm` abstract template class with two arguments, which are specializations of the `Network` and the `VirtualNetworkRequest` template classes. Derivatives of the `EmbeddingAlgorithm` base class implement the `embedVNR` function. This design choice accommodates various VNE approaches described in this paper. For example, a two-stage VNE algorithm may be derived from the `EmbeddingAlgorihm` base class by implementing a two-stage `embedVNR` function that sequentially performs the node and link mappings.

3.3 Discrete Event Simulation Classes

The VNE process is modeled as a discrete event system using the discrete event system specification (DEVS) [19] that has been widely employed for modeling social and ecological systems as well as computer networks and architecture. The framework is implemented using the *Adevs* library [13]. This design enables seamless implementation of various VNE approaches including single and batch processing.

The VNRs are first *generated* based on an arrival process. The arrived VNRs are then queued until they are processed by an embedding algorithm. Hence, the VNRs experience both queuing and processing delays. Based on the outcome of the embedding process, a VNR may either be accepted or rejected. When the life-time of a successfully embedded VNR expires, the *release process* begins and an employed algorithm releases the resources.

The directed acyclic graph (DAG) used for modeling the VNE process in *VNE-Sim* is shown in Figure 2. *VNR Generator*, *Embedding Process*, *Release Process*, and *Statistic Collection* are four modules used to model the VNE process. These components are implemented as abstract template classes that require a specialization of the VNR class as their template argument. The *VNR Generator* models the VNR generation process. Upon the arrival of a VNR, the *VNR Generator* sends a signal to the *Embedding Process* and the *Statistic Collection* modules. The *Embedding Process* is used to model the embedding approach. For example, single and batch VNR processing approaches may be implemented as derivatives of the *Embedding Process* template class. If a VNR is embedded successfully, the *Embedding Process* notifies the *Release Process* and the *Statistic Collection* modules. The *Statistic Collection* implements the Observer design pattern [20]. It is used for logging, collecting statistics, and recording the embedding outcomes. The events that occur in the first three modules are transparent to the *Statistic Collection*. A subscription-based approach is employed in this module where various statistical analysis algorithms may subscribe to receive event updates. A logging subsystem is subscribed to the *Statistic Collection* module by default. The logging subsystem employs the *Hiberlite* library to populate event updates into the SQLite database tables.

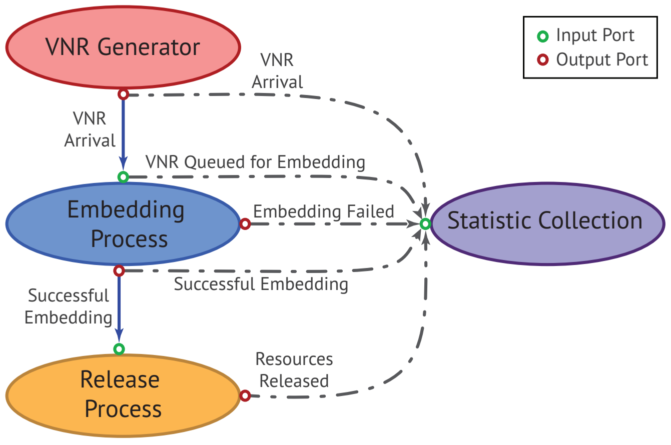

Figure 2: Directed acyclic graph used for modeling the VNE process in *VNE-Sim*. VNRs are first generated and passed to the *Embedding Process*. If a VNR is successfully embedded, it is forwarded to the *Release Process*. The *Statistic Collection* module is an observer that receives information from all input/output ports of other modules.

3.4 Experiment and Result Collection Classes

A *VNE-Sim* simulation scenario is defined using the *Experiment* template class. This class also maintains statistics that are collected during simulations and records them when simulations are completed.

3.5 Operation Related Classes

The `ConfigManager` class reads the simulator configuration file and maintains a *Boost Property Tree* structure of the configurations. This class is a singleton [20] and thus it may be instantiated only once. A pointer to its instance should be acquired first. After obtaining the pointer, a property value may be acquired by calling the `getConfig` function.

The `DBManager` class creates instances of `hiberlite::Database` [12] class and maintains pointers to the created databases in a `map` data structure. This class is also implemented as a singleton. Its interface contains two main functions: `createDB` and `getDB`. The `createDB` function creates a database and returns its pointer. Pointers to the created databases may also be acquired using the `getDB` function.

All network component classes require identification numbers. Classes that have similar types should not possess identical identification numbers. The singleton `IdGenerator` class produces such type-based unique numbers. It generates consecutive identification numbers in an increasing order. The `IdGenerator` interface consists of two template functions: `getId` and `peekId`. The `getId` generates a unique ID for a class type that is given as its template argument while the `peekId` function returns the next ID that will be generated.

The `RNG` class employs the GNU Scientific Library [15]. It is used to generate random numbers and probability distributions. The seed and type of the random number generator are defined in the configuration file. Classes using `RNG` may either use a general purpose random number generator or subscribe to `RNG` class to get a specific generator.

4. SIMULATION OF VNE ALGORITHMS

The VNE problem may be divided into two subproblems: Virtual Node Mapping (VNoM) and Virtual Link Mapping (VLiM). VNoM algorithms map virtual nodes onto substrate nodes while VLiM algorithms map virtual links onto substrate paths. Algorithms that have been proposed for solving VNE are categorized into three categories depending on the approaches taken to solve these two subproblems [21]. The *uncoordinated two-stage algorithms* first solve the VNoM problem and provide a node mapping to the VLiM solver. In this approach, the VNoM and VLiM solvers operate independently without coordination [3]. The *coordinated two-stage algorithms* also first solve the VNoM problem. Unlike the uncoordinated approach, these algorithms consider the virtual link mappings when solving the VNoM problem [4], [6]. The *coordinated one-stage algorithms* solve the VNoM and VLiM problems simultaneously. When two virtual nodes are mapped, also mapped is the virtual link connecting the two nodes.

Most VNE algorithms proposed in the literature address the VNoM while solving the VLiM using the shortest-path algorithms (k-shortest path, Breadth-First Search (BFS), and Dijkstra) or the MCF algorithm [5]. Unlike the MCF algorithm, the shortest-path algorithms do not allow path splitting [3], which enables a virtual link to be mapped onto multiple substrate paths.

4.1 VNE Simulation Performance Metrics

The objective of a VNE algorithm is to maximize an InP's revenue while minimizing the cost of embeddings. The average revenue is directly proportional to the acceptance ratio. Furthermore, increasing the node and link utilizations simultaneously may generate additional revenue. Note that high node utilization is always desirable because it is a result of high acceptance ratio while the link utilization should be maintained as low as possible.

Revenue

Let us denote the available CPU capacity of a node n and available bandwidth of a link e by $\mathcal{C}(n)$ and $\mathcal{B}(e)$, respectively. The revenue $R(G^v)$ is the weighted sum of the VNR's CPU and bandwidth requirements defined as:

$$\mathbf{R}(G^v) = \omega_c \sum_{n^v \in N^v} \mathcal{C}(n^v) + \omega_b \sum_{e^v \in E^v} \mathcal{B}(e^v), \qquad (1)$$

where $G^v(N^v, E^v)$ denotes the virtual network graph while the weights ω_c and ω_b are unit prices of the requested CPU and bandwidth, respectively.

Cost

The cost of a VNE $\mathcal{C}(G^v)$ depends on the allocated substrate resources. It is calculated as:

$$\mathbf{C}(G^v) = \sum_{n^v \in N^v} \mathcal{C}(n^v) + \sum_{e^v \in E^v} \sum_{e^s \in E^s} f_{e^s}^{e^v}, \qquad (2)$$

where $G^s(N^s, E^s)$ denotes the substrate network graph and $f_{e^s}^{e^v}$ is the bandwidth of the substrate link e^s that is allocated to the virtual link e^v.

Acceptance Ratio

The acceptance ratio is calculated as:

$$p_a^\tau = \frac{|\Psi^a(\tau)|}{|\Psi(\tau)|}, \qquad (3)$$

where $|\Psi^a(\tau)|$ and $|\Psi(\tau)|$ denote the number of accepted VNRs and the number of VNRs that arrived in the time interval τ, respectively.

Node and Link Utilizations

Node utilization $\mathcal{U}(N^s)$ is calculated as:

$$\mathcal{U}(N^s) = 1 - \frac{\sum_{n^s \in N^s} \mathcal{C}(n^s)}{\sum_{n^s \in N^s} \mathcal{C}_{max}(n^s)}, \qquad (4)$$

where $\mathcal{C}(n^s)$ is the available CPU resource of a substrate node n^s and $\mathcal{C}_{max}(n^s)$ is the maximum CPU resource of the node. Similarly, link utilization $\mathcal{U}(E^s)$ is calculated as:

$$\mathcal{U}(E^s) = 1 - \frac{\sum_{e^s \in E^s} \mathcal{B}(e^s)}{\sum_{e^s \in e^s} \mathcal{B}_{max}(e^s)}, \qquad (5)$$

where $\mathcal{B}(e^s)$ is the available bandwidth of a substrate link e^s and $\mathcal{B}_{max}(e^s)$ is the maximum bandwidth of the link.

4.2 ViNE and Global Resource Capacity Algorithms

The Global Resource Capacity (GRC) [6], R-ViNE [4], and D-ViNE [4] are examples of coordinated two-stage algorithms. They are implemented in *VNE-Sim* and we employ them for the case study in this paper. The GRC algorithm relies on a node-ranking approach to solve the VNoM subproblem. The rank of a node is calculated based on the CPU capacity of the node and the bandwidth of the incident links [6]. After calculating the ranks of substrate and virtual nodes, the GRC algorithm employs a *large-to-large and small-to-small* [22] mapping scheme to embed virtual nodes onto substrate nodes. After completing the VNoM assignments, a modified Dijkstra's algorithm is used to calculate the shortest path between mapped nodes in order to complete VLiM. The R-ViNE and D-ViNE algorithms formulate the VNoM problem as an MIP. They employ a rounding-based approach to solve the linear programming (LP) relaxation of the proposed MIP. The MCF algorithm is then employed to solve the VLiM subproblem. MCF treats nodes and links of VNRs as commodities and flows, respectively. When solving VLiM, MCF may use path splitting [3]. It has been shown that utilizing MCF results in better utilization of substrate resources [4].

4.3 Simulation Scenarios

During the development phase of *VNE-Sim*, we have performed various test cases and small scale validation scenarios to ensure the correctness of the implemented algorithms. We present here a simulation scenario that is of comparable scale to the scenarios that are presented in the literature [3], [4], [6].

The BRITE [18] library is used to generate the substrate and VNR graphs. The substrate graph is composed of 50

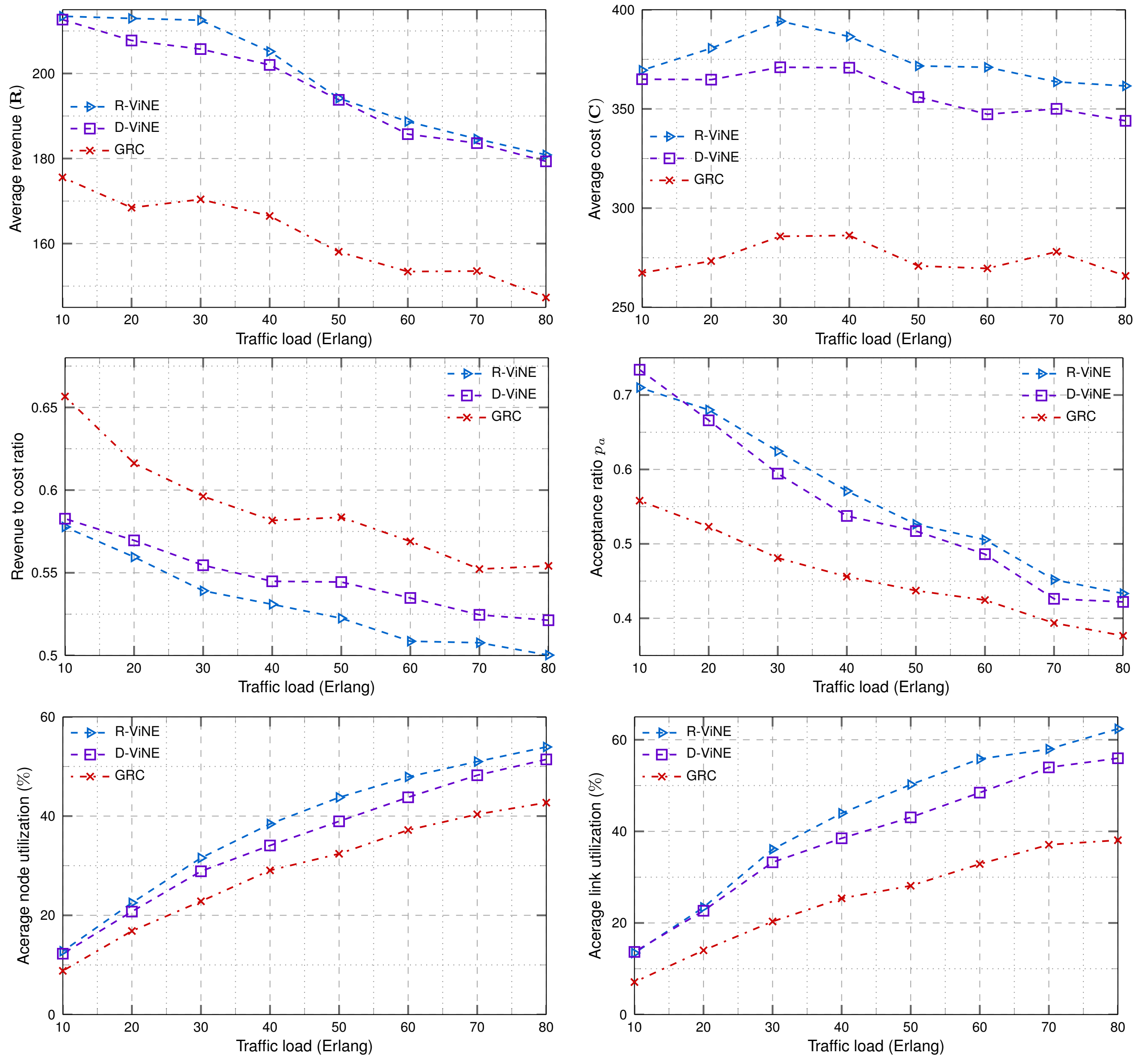

Figure 3: Performance of the ViNE and GRC algorithms. Shown are various performance metrics as functions of the VNR traffic load.

nodes where each node is randomly connected to a maximum of 5 other nodes. The substrate graph that was generated for the simulation scenarios consists of 221 edges. Each node of the substrate network is randomly placed on a 25×25 grid. The VNR graphs are generated using a similar process. The number of nodes in each VNR graph is uniformly distributed between 3 and 10 [4]. The CPU capacity of substrate nodes and the bandwidth of substrate links are uniformly distributed between 50 and 100 units. The CPU requirements of the virtual nodes are uniformly distributed between 2 and 20 units while the bandwidth requirements of the virtual links are uniformly distributed between 0 and 50 units [4]. Performance of the ViNE [4] and the GRC [6] algorithms in terms of revenue, cost, revenue to cost ratio, acceptance ratio, node utilization, and link utilization are shown in Figure 3.

Unprocessed data generated by discrete events that occur during simulations are saved as SQLite databases. These data may be processed using SQL commands or other statistical analysis tools. For example, the output database of the simulation results that are presented here contains a record indicating the outcome of the embedding for every VNR arrival. If a VNR is successfully embedded, the record also includes the revenue, cost, and queuing and processing times of the request. The database entries may then be processed to calculate revenue, cost, and resource utilizations shown in Figure 3.

The R-ViNE and D-ViNE algorithms utilize the substrate link resources more efficiently than the GRC algorithm as a result of employing the MCF algorithm for link mapping. GRC algorithm employs a shortest-path-based algorithm without path splitting to solve VLiM, which is stricter than the MCF algorithm that enables path splitting. For example, consider a virtual node attached to a virtual link that requires 5 units of bandwidth and a substrate node attached to two substrate links with available bandwidths of 2 and 3 units. In this case, while utilizing MCF permits embedding the virtual node onto the substrate node, such embedding is infeasible without path splitting. Superior performance of the R-ViNE and D-ViNE algorithms comes at the cost of higher execution time. In the simulation scenarios, a virtual network embedding required on average 0.596 s, 0.591 s, and 0.017 s using the R-ViNE, D-ViNE, and GRC algorithms, respectively.

VNE-Sim has been used to compare the BCube$(2, 4)$ [23] and Fat-Tree$_6$ [24] data center network topologies for virtual network embedding by employing the R-ViNE, D-ViNE, and GRC algorithms [9]. Simulation results indicate that the Fat-Tree topology is capable of accepting additional number of virtual network requests even though the topology consists of slightly fewer number of nodes and links.

5. CONCLUSION

In this paper, we presented *VNE-Sim* for simulating VNE algorithms. *VNE-Sim* formalizes the VNE process as a discrete event system based on the DEVS framework and uses the *Adevs* library. The modular and scalable design of *VNE-Sim* enables researchers to seamlessly implement new VNE algorithms and evaluate their performance.

The initial development phase of *VNE-Sim* has been completed. Future work includes implementing other VNE algorithms, implementing a source code documentation system, in depth comparison of *VNE-Sim* and *Alevin* in terms of memory usage and capabilities, and implementing a scripting infrastructure for visualization of results.

6. REFERENCES

[1] J. S. Turner and D. E. Taylor, "Diversifying the Internet," in *Proc. IEEE GLOBECOM 2005*, St. Louis, MO, USA, Dec. 2005, pp. 755–760.

[2] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, Jan. 2007.

[3] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *Comput. Commun. Rev.*, vol. 38, no. 2, pp. 19–29, Mar. 2008.

[4] M. Chowdhury, M. R. Rahman, and R. Boutaba, "ViNEYard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[5] D. G. Andersen, "Theoretical approaches to node assignment," Dec. 2002, Unpublished Manuscript. [Online]. Available: http://repository.cmu.edu/compsci/86/.

[6] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *Proc. IEEE INFOCOM*, Toronto, ON, Canada, Apr. 2014, pp. 1–9.

[7] M. T. Beck, C. Linnhoff-Popien, A. Fischer, F. Kokot, and H. de Meer, "A simulation framework for virtual network embedding algorithms," in *Proc. 16th Int. Telecommunications Netw. Strategy and Planning Symp.*, Funchal, Portugal, Sept. 2014, pp. 1–6.

[8] S. Haeri, Q. Ding, Z. Li, and Lj. Trajković, "Global resource capacity algorithm with path splitting for virtual network embedding," in *Proc. IEEE Int. Symp. Circuits and Systems*, Montreal, QC, Canada, May 2016, pp. 666–669.

[9] S. Haeri and Lj. Trajković, "Virtual network embeddings in data center networks," in *Proc. IEEE Int. Symp. Circuits and Systems*, Montreal, QC, Canada, May 2016, pp. 874–877.

[10] (2015, Dec.) CMake Build System. [Online]. Available: https://cmake.org/.

[11] (2015, Dec.) Fast Network Simulation Setup. [Online]. Available: http://fnss.github.io/.

[12] (2015, Dec.) Hiberlite Library. [Online]. Available: https://github.com/paulftw/hiberlite/.

[13] J. J. Nutaro, *Building Software for Simulation: Theory and Algorithms, with Applications in C++*. Hoboken, NJ, USA: John Wiley & Sons, 2010.

[14] (2015, Dec.) Boost C++ Libraries. [Online]. Available: http://www.boost.org/.

[15] (2016, Jan.) GSL–GNU Scientific Library. [Online]. Available: https://www.gnu.org/software/gsl/.

[16] (2016, Jan.) GLPK–GNU Linear Programming Kit. [Online]. Available: http://www.gnu.org/software/glpk/.

[17] (2015, Dec.) SQLite: Small. Fast. Reliable. Choose any three. [Online]. Available: https://www.sqlite.org/.

[18] (2016, Jan.) Boston University Representative Internet Topology Generator. [Online]. Available: http://www.cs.bu.edu/brite/.

[19] A. M. Uhrmacher, "Dynamic structures in modeling and simulation: a reflective approach," *ACM Trans. Modeling and Computer Simulation*, vol. 11, no. 2, pp. 206–232, Apr. 2001.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., ser. Addison-Wesley Professional Computing Series. Boston, MA, USA: Addison-Wesley, 1994.

[21] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[22] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," *Comput. Commun. Rev.*, vol. 41, pp. 38–47, Apr. 2011.

[23] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: a high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, Oct. 2009.

[24] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Oct. 2008.